February 23, 2005

NIST GSC-IS V2.1 Conformance Test Suite Maintenance Manual

http://smartcard.nist.gov

This document consists of seven parts that are useful for the maintenance of GSC-IS V2.1 conformance test suites.

TABLE OF CONTENTS

How to Create Test Scenarios in XML

This part presents the rules and structures for translating narrative test scenarios into the XML format. It applies to both the BSI and Card Edge interfaces.

The Conformance Test Instantiation, Verification and Report Scenarios

The four conformance test instantiation, verification and report scenarios documents by Alan Goldfine are used as the basis for coding XML. There is a one to one correspondence between the narrative scenarios and the XML scenarios. Several test scenarios are associated with each BSI command or Card Edge APDU. The Test for Assertion numbers identified in the Scenarios document corresponds to the XML tag "Assertions id." Each Test for Assertion may contain one or more pre-condition command calls, an instantiation call (the actual test call for the command), and one or more verification command calls. For the BSI, the "Extended Error Text" call is inserted after the appropriate test sequence. For the APDU, a "Get Response" call may be inserted as appropriate.

The DTD File

The XML coding must be "well defined" with respect to a Document Type Definition (DTD). The W3C Recommendations for writing the XML and DTD files are used. The DTD is organized as a tree structure. There is one DTD for the BSI Java, one for the BSI C-Binding, and one for both the file system and virtual machine. Card Edge.

The BSI Java Binding DTD is named "jsdtd.dtd." The root is called "**test-method**"

The BSI C Binding DTD is named "csdtd.dtd." The root is called "**test-function**"

The Card Edge Interface DTD is named "CE.dtd." The root is called "**test-APDU**"

The following Figure 1 represents the DTD for the BSI Java Binding as a W3C Schema.

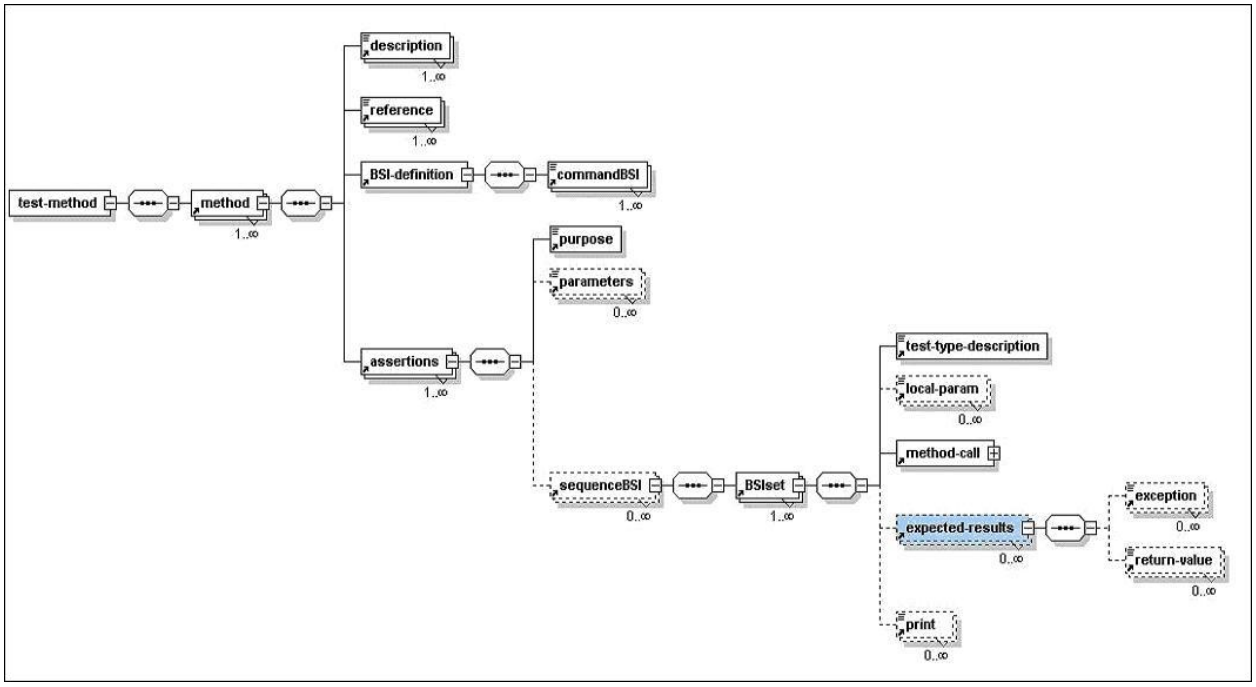


Figure 1: W3C Schema of the BSI Java DTD file.

The tags and their associated attributes are defined in the DTD file.

The XML File

There are 23 XML files for each of the BSI Java binding and the BSI C binding. There are 13 XML files for the Card Edge file system cards, and 12 XML files for the Card Edge virtual machine cards.

Each XML file consists of a sequence of BSI or APDU test cases. The test case number corresponds to the Test for Assertion number in the corresponding scenarios document.

For each test case, the XML code consists of the command call including parameters, the expected results, and the print statement that indicates Pass/Fail/Feature-Not-Supported.

The following Figure 2 presents the block structure of an XML file used to test a BSI Java method.

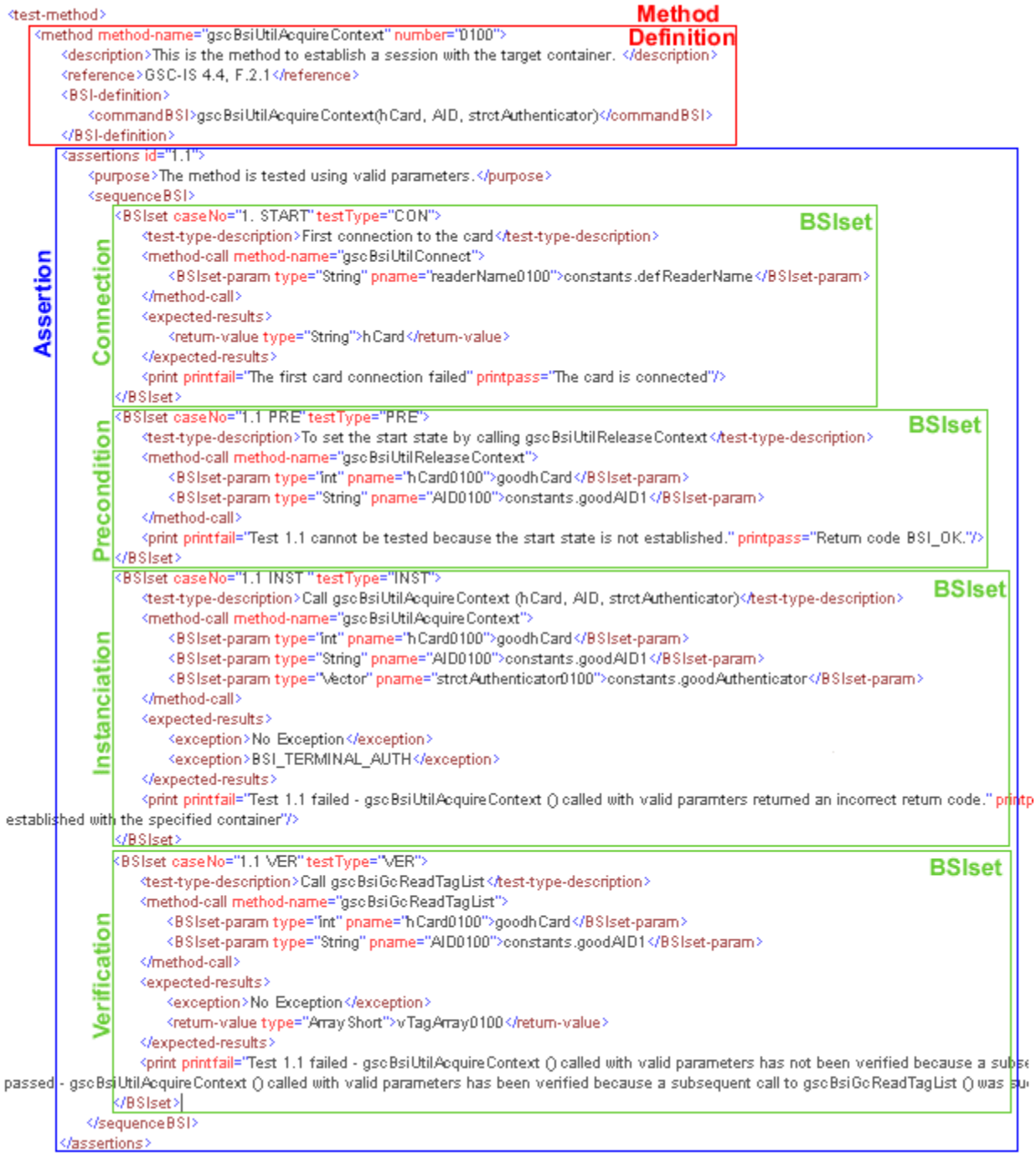


Figure 2: Structure of the BSI Java XML file.

Each test for assertion is composed of a series of BSI calls (i.e., the element BSIset), which may be a precondition call (PRE), an instantiation call (INST), an extended error text call (TEXT), a verification call (VER) or another type call (OTH). The element BSIset gathers all the information needed to issue the BSI call and to handle the response code and return parameters. The BSI call itself is defined within the tag as follows:

“<method-call method-name = "gscBsiUtilAcquireContext">”.
The name of the called method is defined by the attached attribute “method-name”.

The Expected Results

Each command call may throw an exception (e.g., response code). A print statement can be associated with any particular exception value, coded as an attribute to the exception tag. For example:

```
...
<exception messageNT="No Card Service">BSI_NO_CARDSERVICE</exception>
...
```

Here, the print statement named with the messageNT attribute would execute if the exception BSI_NO_CARDSERVICE is thrown by the implementation under test.

If there is no expected exception for the command, and none occurs, the content of the “printpass” attribute is executed. If an exception does occur, the content of the “printfail” is executed (This is the default behavior). If one of the expected exceptions occurs and if it has its own print statement (as messageF, messageP or messageNT), the associated message will be displayed instead of the normal printpass/printfail.

If a specific print statement needs to be executed for a “No Exception" case, it should appear as a “No Exception” statement. The following is an example:

```
…
<exception messageF="Test 14.1 failed - gscBsiGcDataDelete() called with valid parameters has not been verified because a subsequent call to gscBsiGcReadValue() indicated that the specified data item was not deleted.">No Exception</exception>
…
```

The Attributes Associated with the Exceptions: messageF, messageP, messageNT

The XML statements have certain attributes associated with the evaluation of the test results. They are defined in the DTD file as:

“messageP” : associated with an exception which evaluates the test as “passed”.
“messageF”: associated with an exception which evaluates the test as “failed”.
“messageNT”: associated with an exception which evaluates the test as “not testable”.

The Manual Tasks

For the BSI tests, the XML statements also define attributes associated with instructions to the operator to perform such manual tasks as removing the inserted card, or inserting a bad card. They are defined in the DTD file as, for example:

“opMode = PAUSE”: the test will pause with a pop-up window asking the operator to perform the manual task: REMOVE CARD.

Criteria to Determine the Outcome of a Test Case

There are three possible outcomes for a conformance test case:

1. "**Passed**," shown in green in the test output
2. "**Failed**," shown in red
3. "**Unable to be Tested**," shown in yellow

The following is an attempt to define these outcomes further and to catalog the sets of criteria for each of them.

A scenario is considered to have **Passed** if:

- all starting state and instantiation scenario "Pre" conditions are satisfied, and
- all verification goal conditions are satisfied.

A scenario is considered to have **Failed** if:

- all starting state and instantiation scenario "Pre" conditions are satisfied, and
- at least one verification goal conditions is not satisfied.

A scenario is considered to be **Unable to be Tested** if either

- at least one starting state or instantiation scenario "Pre" condition is not satisfied, or
- the evaluation of at least one verification goal condition is ambiguous. Some examples of this situation are BSI command calls that return BSI_NO_CARDSERVICE or BSI_NO_SPSSERVICE, and card edge READ BUFFER APDUs attempting to verify the contents of a container after an UPDATE BUFFER.

The determination of whether or not a condition is satisfied involves an examination of the codes and values returned by the command call associated with that condition. For the BSI C binding, all command calls return a return code. In the BSI Java binding, the "No Exception" is the equivalent of the BSI_OK return code as in the C binding. Some of the commands also return a return value. The generic determination procedure includes several cases and subcases.

Case 1: The return code equals the expected return code.

Case 1.1: There is no return value. **The condition is satisfied.**

Case 1.2: There is a return value.

Case 1.2.1: There is an expected value for the return value.

Case 1.2.1.1: The return value equals the expected value. **The condition is satisfied.**

Case 1.2.1.2: The return value does not equal the expected value. **The condition is not satisfied.**

Case 1.2.2: There is no expected value for the return value. This situation occurs in cases where we have some general idea of the format and allowable values of the return value, but not its exact expected value. Examples include the BSI implementation version in gscBsiUtilGetVersion, and the challenge generated by the smart card in gscBsiGetChallenge. In these cases the program is paused for a manual inspection of the return value by the tester.

Case 1.2.2.1: The tester responds that the return value is correct. The program resumes and **the condition is satisfied.**

Case 1.2.2.2: The tester responds that the return value is not correct. The program resumes and **the condition is not satisfied.**

Case 2: The return code does not equal the expected return code. **The condition is not satisfied.**

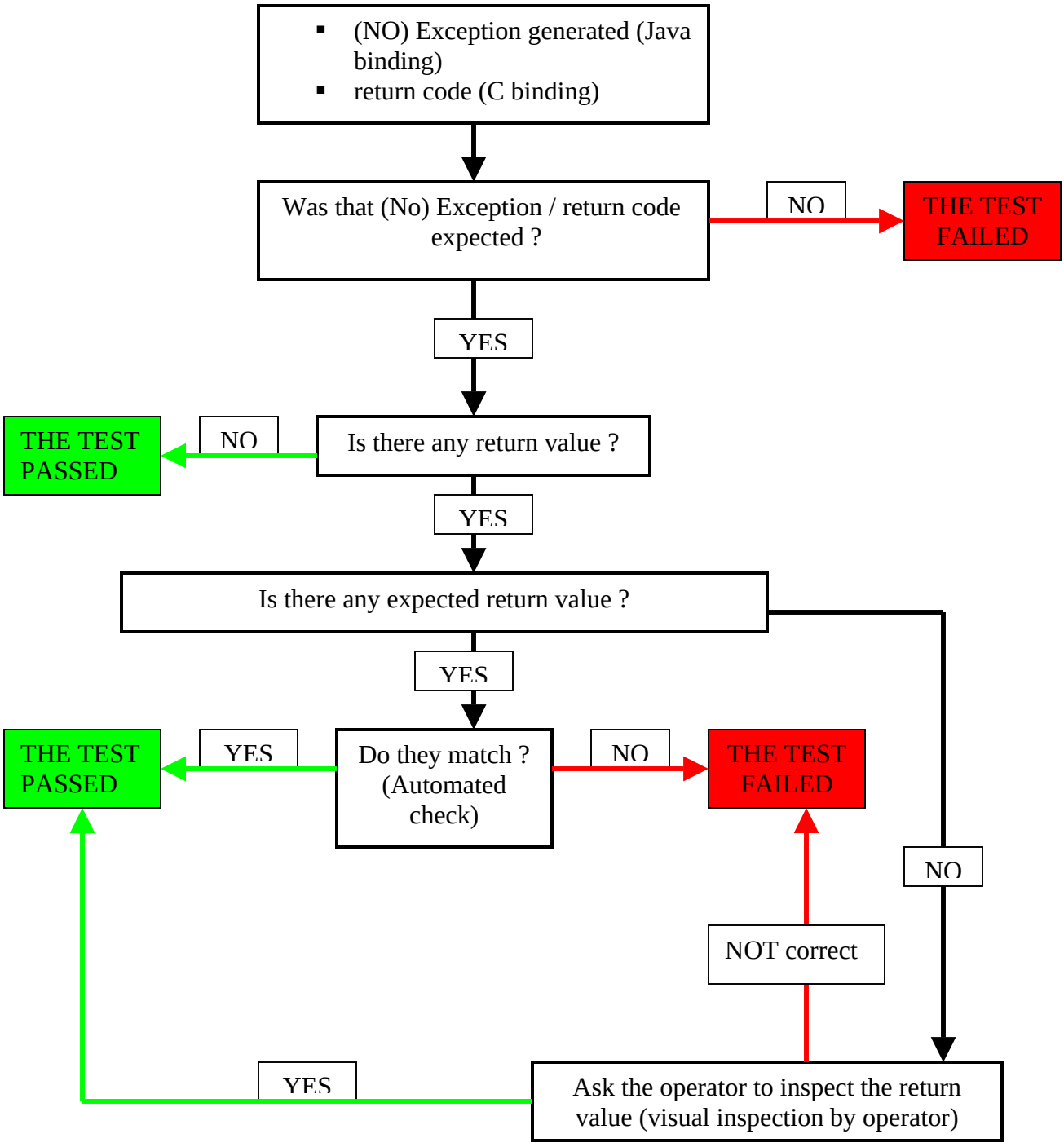


Figure 3: Diagram showing the logic which defines the status of one single test case.

Customization of the Symbolic Constants and Path

The values of parameters in each of the test suites are abstracted by the use of symbolic constants, defined in Appendix A of the Scenarios documents of the respective test suite. The actual values of these symbolic constants are contained in a constants file. Therefore, their values may be customized by editing the constants files.

The constant file for the BSI tests is
src\gov\nist\smartcard\gscis\testing\BSIconstants.java

The constant file for the File System and Virtual Machine Card Edge Interface test is
src\gov\smartcard\gscis\testing\CEconstants.java.

There are 2 generic variables for the BSI tests that may be customized:

- goodAID1 and goodAID2 (See **src\gov\nist\smartcard\gscis\testing\BSIconstants.java**)

BSI and Card Edge Path Settings

The TEST_HOME path that contains path variables for the test results and log file directories may be customized by editing the file (See
src\gov\nist\smartcard\gscis\testing\pathSettings.java)

List of Cards Required for Testing of BSI Implementations

See Appendix B of the Test Scenarios Documents (C, Java, Card Edge FS, Card Edge VM).

Important note:

A BSI implementation can only be tested if smart cards conformant with the GSC-IS 2.1 are available. Those conformant smart cards would presumably already have passed the appropriate Card Edge conformance testing.

GSC Conformance Test Interpretations, Assumptions, and Test Design Tricks

There are several known ambiguities in the GSC-IS v2.1 specification. These ambiguities can lead to varying interpretations and assumptions by product vendors and by designers of conformance tests.

This part lists some of the assumptions made by the conformance test developers, and discusses how the conformance test suites handle them.

In the design of test scenarios, not all combinations and permutations cases were developed. Some tests are designed to cover certain situations and these are described here.

NULL vs " "

In gscUtilCardConnect (Java BSI), if readerName = " ", it is valid and the command should connect to the next available reader.
In gscUtilCardConnect (C BSI), if readerName = “NULL” it is valid and the command should connect to the next available reader.

BAD_PARAM

In Java, the “BSI_BAD_PARAM” return code is assumed to result from the use of a parameter with incorrect content.

In C, the “BSI_BAD_PARAM” return code is generated by the use of a length parameter that incorrectly specifies the length of the associated parameter.

In both C and Java, “BSI_BAD_TAG return code means a non-existent tag, rather than a syntactically incorrect one. If one of the parameter is invalid (syntactically incorrect), the code “BSI_BAD_PARAM” is generated.

BSI_PIN_BLOCK

In Test 1.8 (Java) and 1.9 (C), the test attempts to generate the exception BSI_PIN_BLOCK. We assume that there exists a number N representing the maximum number of tries allowed by the implementation. The conformance test program executes the gscBsiUtilAcquireContext function N times with a badPinAuthenticator parameter, and expects that each time the exception code will be “BSI_BAD_AUTH”. The program will then execute the command once more, this time expecting BSI_PIN_BLOCKED.

Discovery Methods in C used to test a bad handle (or bad version length)

In C-binding functions with discovery methods, we designed two separate test cases to test those functions with a bad handle (or a bad version length):

(a) Discovery Method 1, Discovery Mode call with bad handle (or bad version), if we do get the expected return code (BSI_BAD_HANDLE) then the test ends.

(b) Discovery Method 1, Discovery Mode first call with good handle (good version) to obtain the correct length. We then do the Discovery Method 1, Final Mode call with a bad handle (or bad version) with the obtained length. If we do get the expected return code, than the test ends.

AcquireContext Test to set ACR requiring (XAUTH)

In Test 1.4 (Java) and Test 1.5 (C), we issue the GetChallenge() call before the AcquireContext() call for the following reason:

In the GSC-IS, Section 4.5.3, gscBsiUtilAcquireContext, the purpose paragraph states "For ACRs requiring external authentication (XAUTH), the authValue field of the BSIAuthenticator structure must contain a cryptogram calculated by encrypting a random challenge from gscBsiGetChallenge()."

Therefore, it is possible that an implementation may require a GetChallenge() call before a AcquireContext() call in such a situation. To take care of this possibility, we issue the GetChallenge().

It should be noted that the actual content of the challenge that is placed in the BSIAuthenticator is unimportant.

REMOVAL OF CARD

(Java and C): We consider the state of a system following the removal of a connected card to be implementer defined, and not subject to further testing, including the testing of GetExtendedErrorText().

gscBsiUtilGetExtendedErrorText

(Java and C): We assume that there is no requirement that gscBsiUtilGetExtendedErrorText() be capable of providing extended messages for the "successful" return codes BSI_OK or BSI_TERMINAL_AUTH.

BSI_NO_CARDSERVICE

(C): BSI_NO_CARDSERVICE is a possible return code for the Discovery Mode of GetCardProperties().

Encrypted Challenge (Data field computed at Runtime)

The Data APDU field of some APDU commands may be computed at runtime. For instance the command EXTERNAL AUTHENTICATE should send an encrypted challenge. The challenge is provided by a previously called GET CHALLENGE APDU command. In the XML, the data field of the corresponding encrypted challenge would have the String "**encrypted challenge**" (NB: This is case sensitive). See XML files for EXTERNAL AUTHENTICATE APDU command (test case 8.1 instantiation for file system type and test case 10.1 instantiation for virtual machine type).

Assumptions made in Card Edge Tests

- (File System and VM Card Edge): We assume that GET RESPONSE is only used in environments using the T=0 communications protocol.
- (File System Card Edge): For READ BINARY, SELECT DF, SELECT EF UNDER SELECTED DF, SELECT FILE, SELECT MASTER FILE, INTERNAL AUTHENTICATE, and PERFORM SECURITY OPERATION in implementations using the T=0 communications protocol, the 61 XX response is required by ISO 7816-4.
- (File System Card Edge): For SELECT DF, SELECT EF UNDER SELECTED DF, SELECT FILE, and SELECT MASTER FILE, we allow for 61 XX even if no response is requested, because some systems may nevertheless generate this response.
- (File System Card Edge): For SELECT DF and SELECT EF UNDER SELECTED DF where P2 == 0C, we assume that L_e is empty.
- (File System Card Edge): For SELECT FILE and SELECT MASTER FILE where P2 == 00, we assume that L_e contains the maximum number of bytes expected in the data field of the response to the command.
- (File System Card Edge): For GET RESPONSE, we assume that if the SELECT MASTER FILE in the Starting State returns SW1 SW2 == 61 LL, then LL < FF.
- (File System and VM Card Edge): For EXTERNAL AUTHENTICATE, we interpret the situation "Conditions of use not satisfied (the command is not allowed in this context)" to mean that EXTERNAL AUTHENTICATE must immediately follow a GET CHALLENGE.
- (File System and VM Card Edge): The current test suite does not provide test for the following conditions:
    a. Check selected file is deactived.

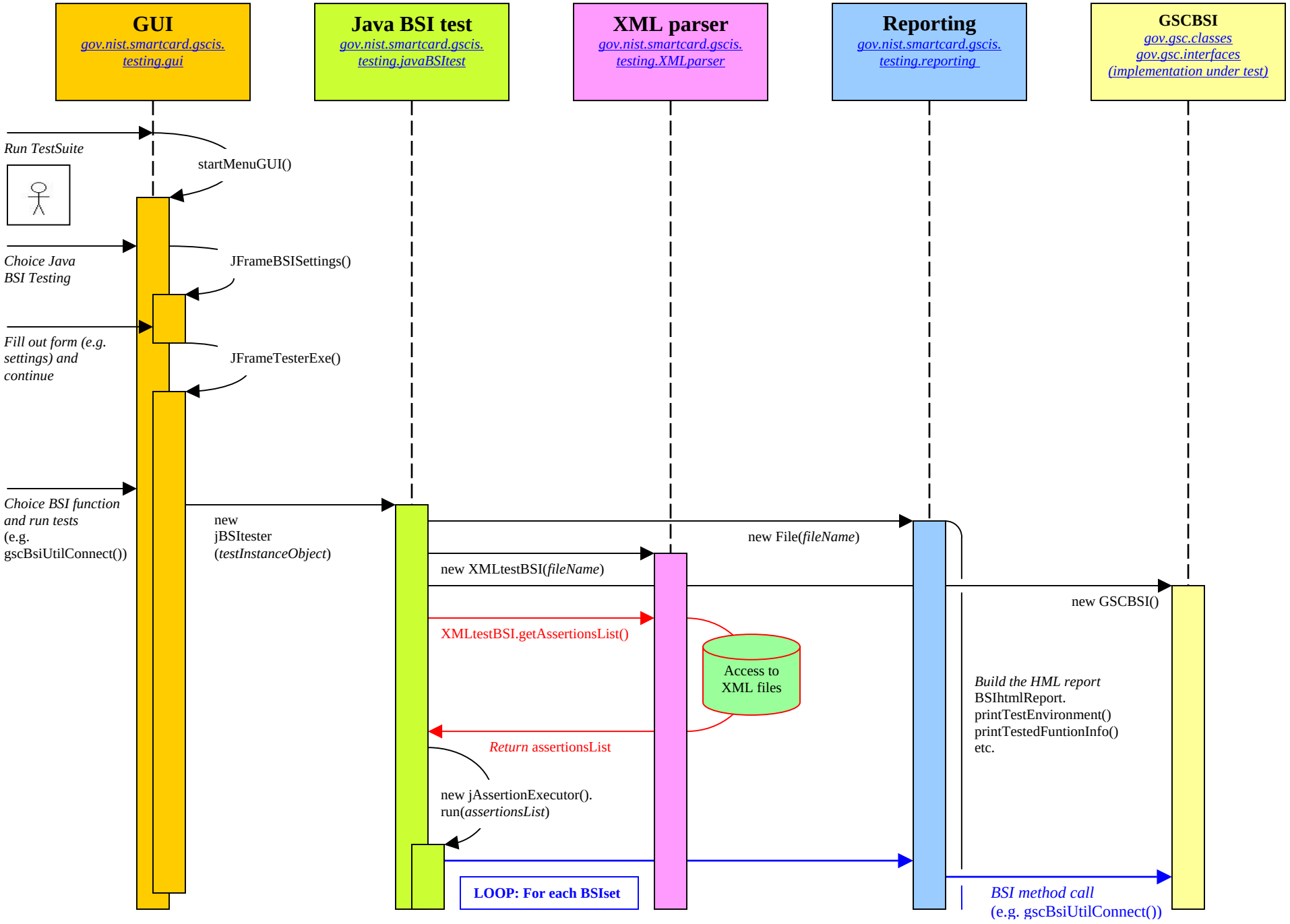
b. Check that the card which FCI is not formatted according to ISO 7816-4 Section 5.1.5.

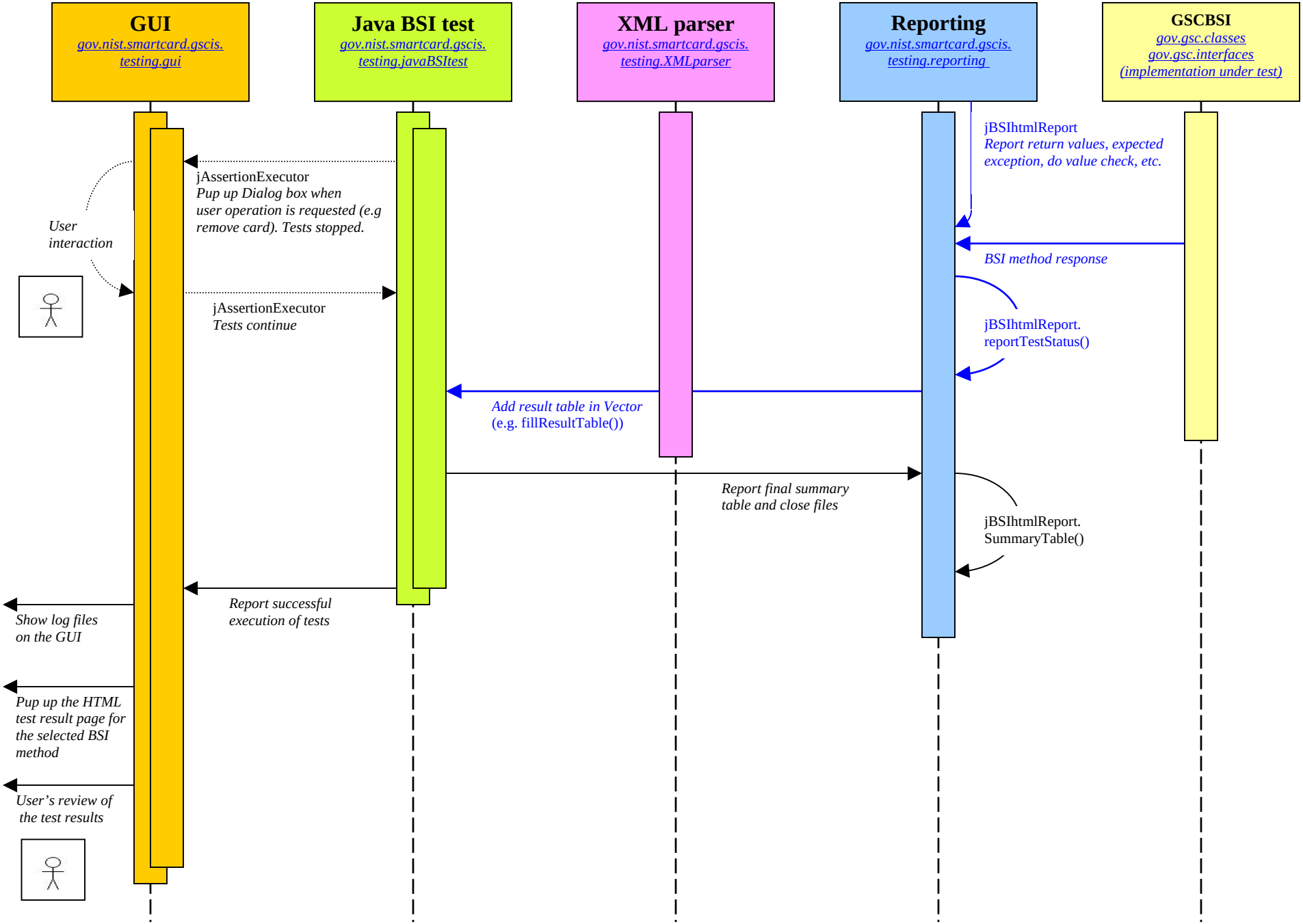
UML Diagram for the GSC-IS 2.1

The GSC-IS v2.1 conformance test suite is composed of the following Java packages:

gov.gsc.classes	Classes in GSC-IS v2.1
gov.gsc.interfaces	Interfaces in GSC-IS v2.1
gov.nist.smartcard.gscis.testing	Test suite package
gov.nist.smartcard.gscis.testing.CardEdgeTest	Card Edge testing classes
gov.nist.smartcard.gscis.testing.cBSItest	C BSI Binding testing
gov.nist.smartcard.gscis.testing.gui	GUI classes
gov.nist.smartcard.gscis.testing.javaBSItest	Java BSI Binding testing
gov.nist.smartcard.gscis.testing.reporting	Reporting classes
gov.nist.smartcard.gscis.testing.util	Utilities classes
gov.nist.smartcard.gscis.testing.XMLparser	XML parsing classes

The following UML sequence diagram shows the tests of a Java BSI implementation


GUI
gov.nist.smartcard.gscis.testing.gui
Java BSI test
gov.nist.smartcard.gscis.testing.javaBSItest
XML parser
gov.nist.smartcard.gscis.testing.XMLparser
Reporting
gov.nist.smartcard.gscis.testing.reporting
GSCBSI
gov.gsc.classes
gov.gsc.interfaces
(implementation under test)
Run TestSuite
startMenuGUI()
Choice Java BSI Testing
JFrameBSISettings()
Fill out form (e.g. settings) and continue
JFrameTesterExe()
Choice BSI function and run tests (e.g. gscBsiUtilConnect())
new jBSItester (testInstanceObject)
new File(fileName)
new XMLtestBSI(fileName)
new GSCBSI()
XMLtestBSI.getAssertionsList()
Access to XML files
Build the HML report
BSIhtmlReport.
printTestEnvironment()
printTestedFuntionInfo()
etc.
Return assertionsList
new jAssertionExecutor().
run(assertionsList)
LOOP: For each BSIset
BSI method call
(e.g. gscBsiUtilConnect())


GUI
gov.nist.smartcard.gscis.testing.gui
Java BSI test
gov.nist.smartcard.gscis.testing.javaBSItest
XML parser
gov.nist.smartcard.gscis.testing.XMLparser
Reporting
gov.nist.smartcard.gscis.testing.reporting
GSCBSI
gov.gsc.classes
gov.gsc.interfaces
(implementation under test)
jBSIhtmlReport
Report return values, expected exception, do value check, etc.
User interaction
jAssertionExecutor
Pup up Dialog box when user operation is requested (e.g remove card). Tests stopped.
BSI method response
jAssertionExecutor
Tests continue
jBSIhtmlReport.
reportTestStatus()
Add result table in Vector
(e.g. fillResultTable())
Report final summary table and close files
jBSIhtmlReport.
SummaryTable()
Report successful execution of tests
Show log files on the GUI
Pup up the HTML test result page for the selected BSI method
User's review of the test results